Oracle

1Z0-804 Exam

Java SE 7 Programmer II

Thank you for Downloading 1Z0-804 exam PDF Demo

You can Buy Latest 1Z0-804 Full Version Download

https://www.certkillers.net/Exam/1Z0-804

Question: 1

Given the code fragment:
DataFormat df;
Which statement defines a new Dateformat object that displays the default date format for the UK Locale?

A. df = DateFormat.getdatDataInstance (DateFormat.DEFAULT, Locale (UK));
B. df = DateFormat.getdatDataInstance (DateFormat.DEFAULT, UK);
C. df = DateFormat.getdatDataInstance (DateFormat.DEFAULT, Locale.UK);
D. df = new DateFormat.getdatDataInstance (DateFormat.DEFAULT, Locale.UK);
E. df = new DateFormat.getdatDataInstance (DateFormat.DEFAULT, Locale (UK));

Answer: C

Explanation:
The UK locale is constructed with Locale.UK.

Example:
To format a date for a different Locale, specify it in the call to getDateInstance().
DateFormat df = DateFormat.getDateInstance(DateFormat.LONG, Locale.FRANCE);
Note: getDateInstance(int style, Locale aLocale)
Gets the date formatter with the given formatting style for the given locale.
Reference: Class DateFormat

Question: 2

Given:
```
public class DoubleThread {
public static void main(String[] args) {
Thread t1 = new Thread() {
public void run() {
System.out.print("Greeting");
}
};
Thread t2 = new Thread(t1); // Line 9
t2.run();
}
}
```
Which two are true?

A. A runtime exception is thrown on line 9.
B. No output is produced.
C. Greeting is printed once.
D. Greeting is printed twice.
E. No new threads of execution are started within the main method.

F. One new thread of execution is started within the main method.
G. Two new threads of execution are started within the main method.

Answer: C, E

Explanation:
Thread t2 is executed. Execution of T2 starts executionen of t1. Greeting is printed during the execution of t1.

Question: 3

Given:
```
import java.util.*;
public class AccessTest {
public static void main(String[] args) {
Thread t1 = new Thread(new WorkerThread());
Thread t2 = new Thread(new WorkerThread());
t1.start(); t2.start; // line1
}
}
class WorkPool {
static ArrayList<Integer> list = new ArrayList<>(); // line2
public static void addItem() { // line3
list.add(1); // Line4
}
}
class WorkerThread implements Runnable {
static Object bar = new Object ();
public void run() { //line5
for (int i=0; i<5000;i++) WorkPool.addItem(); // line6
}
}
```
Which of the four are valid modifications to synchronize access to the valid list between threads t1 and t2?

A. Replace line 1 with:
Synchronized (t2) (t1.start();) synchronized(t1) (t2.start();)
B. Replace Line 2 with:
static CopyWriteArrayList<Integer> list = new CopyWriteArrayList<>();
C. Replace line 3 with:
synchronized public static void addItem () {
D. Replace line 4 with:
synchronized (list) (list.add(1);)
E. Replace line 5 with:
Synchronized public void run () {
F. replace line 6 with:
Synchronized (this) {for (in i = 0, i<5000, i++) WorkPool.addItem(); }
G. Replace line 6 with:

synchronized (bar) {for (int i= 0; i<5000; i++) WorkPool.addItem(); }

Answer: F

Explanation:
A way to create synchronized code is with synchronized statements. Unlike synchronized methods, synchronized statements must specify the object that provides the intrinsic lock:
For example:

```
public void addName(String name) {
   synchronized(this) {
      lastName = name;
      nameCount++;
   }
   nameList.add(name);
}
```

In this example, the addName method needs to synchronize changes to lastName and nameCount, but also needs to avoid synchronizing invocations of other objects' methods. Without synchronized statements, there would have to be a separate, unsynchronized method for the sole purpose of invoking nameList.add.
Reference: The Java Tutorial, Intrinsic Locks and Synchronization

Question: 4

Sam has designed an application. It segregates tasks that are critical and executed frequently from tasks that are non critical and executed less frequently. He has prioritized these tasks based on their criticality and frequency of execution. After close scrutiny, he finds that the tasks designed to be non critical are rarely getting executed. From what kind of problem is the application suffering?

A. race condition
B. starvation
C. deadlock
D. livelock

Answer: C

Explanation:
Starvation describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.
Incorrect answers:
B: Deadlock describes a situation where two or more threads are blocked forever, waiting for each other.
D: A thread often acts in response to the action of another thread. If the other thread's action is also a response to the action of another thread, then livelock may result. As with deadlock, livelocked threads are unable to make further progress. However, the threads are not blocked — they are

simply too busy responding to each other to resume work. This is comparable to two people attempting to pass each other in a corridor: Alphonse moves to his left to let Gaston pass, while Gaston moves to his right to let Alphonse pass. Seeing that they are still blocking each other, Alphone moves to his right, while Gaston moves to his left. They're still blocking each other, so...
Reference: The Java Tutorial, Starvation and Livelock

Question: 5

Give:

```
Class Employee {
        public int checkEmail() {/* . . . */}
        public void sendEmail (String email) {/* . . . */}
        public Boolean validDateEmail(){/* . . . */}
        public void printLetter (String letter) {/* . . . */}
}
```

Which is correct?

A. Employee takes advantage of composition.
B. Employee "has-an" Email.
C. Employee "is-a" LetterPrinter.
D. Employee has low cohesion.

Answer: D

Explanation:
The relationship between Employee and e-mail is poorly implemented here.
There is low cohesion.
Note:
Low cohesion is associated with undesirable traits such as being difficult to maintain, difficult to test, difficult to reuse, and even difficult to understand.
Cohesion is decreased if:
The functionalities embedded in a class, accessed through its methods, have little in common.
Methods carry out many varied activities, often using coarsely-grained or unrelated sets of data.
Disadvantages of low cohesion (or "weak cohesion") are:
Increased difficulty in understanding modules.
Increased difficulty in maintaining a system, because logical changes in the domain affect multiple modules, and because changes in one module require changes in related modules.
Increased difficulty in reusing a module because most applications won't need the random set of operations provided by a module.
Reference: Cohesion (computer science)

Question: 6

Which two demonstrate the valid usage of the keyword synchronized?

A. interface ThreadSafe {
synchronized void doIt();

```
}
B. abstract class ThreadSafe {
synchronized abstract void doIt();
}
C. class ThreadSafe {
synchronized static void soIt () {}
}
D. enum ThreadSafe {
ONE, TWO, Three;
Synchronized final void doIt () {}
}
```

Answer: C

Explanation:
The Java programming language provides two basic synchronization idioms: synchronized methods and synchronized statements.
To make a method synchronized, simply add the synchronized keyword to its declaration.
Incorrect answers:
A: Should not use synchronized within an interface.
B: A thread which enters the synchronized method must get the lock of the object (or of the class) in which the method is defined. You can not instantiate an abstract class so there is no object with the lock.
D: Should not use synchronized within an enumeration.

Question: 7

Given the incomplete pseudo-code for a fork/join framework application:

```
submit(Data) {
if(Data.size < SMALL_ENOUGH) {
_____(Data);  // line x
}
else {
List<Data> x = _____(Data); // line Y
for(Data d:  x
_____(d);  // line z
}
}
```

And given the missing methods:
process, submit, and splitInHalf
Which three insertions properly complete the pseudo-code?

A. Insert submit at line X.
B. Insert splitInHalf at line X.
C. Insert process at line X.
D. Insert process at line Y.
E. Insert splitInHalf at line Y.
F. Insert process at line Z.

G. Insert submit at line Z.

Answer: C, E, G

Explanation:
C: If data is small enough then process it. Line X
E: If data is not small enough then split it half. Line Y
G: After the data has been split (line Y) then recursively submit the splitted data (Line z).

Question: 8

ITEM Table
* ID, INTEGER: PK
* DESCRIP, VARCHAR(100)
* PRICE, REAL
* QUALITY, INTEGER
And given the code fragment (assuming that the SQL query is valid):

```
try {
String query = "SELECT * FROM Item WHERE ID=110";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(query);
while (rs.next ()) {
System.out.println("ID:  " + rs.getInt("Id"));
System.out.println("Description:  " + rs.getString("Descrip"));
System.out.println("Price:  " + rs.getDouble("Price"));
System.out.println("Quantity:  " + rs.getInt("Quantity"));
}
} catch (SQLException se) {
System.out.println("Error");
}
```

What is the result of compiling and executing this code?

A. An exception is thrown at runtime
B. Compile fails
C. The code prints Error
D. The code prints information about Item 110

Answer: A

Explanation:
The connection conn is not defined. The code will not compile.

Question: 9

Given:

```
class Deeper {
        public Number getDepth() {
```

return 10;
}
}
Which two classes correctly override the getDepth method?

A. public class deep extends Deeper {
protected integer getDepth(){
return 5;
}
}
B. public class deep extends Deeper {
public double getDepth() {
return "5";
}
}
C. public class deep extends Deeper {
public String getDepth () {
}
}
D. public class deep extends Deeper {
public Long getDepth (int d) {
return 5L;
}
}
E. public class deep extends Deeper {
public short getDepth () {
return 5;
}
}

Answer: A, E

Explanation:
Note: The abstract class Number is the superclass of classes Byte, Double, Float, Integer, Long, and Short. Subclasses of Number must provide methods to convert the represented numeric value to byte, double, float, int, long, and short.
When class C extends B, we say that C is a "subclass" of B, and B is the "superclass" of C. This is called inheritence, because C inherited from B.
Incorrect answers:
B: A letter is returned.
C: The String data type cannot be used to extend Number.
D: We should not add a parameter to the extended class.
public Long getDepth (int d)

Question: 10

Given the code fragment:
public class App {

```
public static void main (String [] args){
                    Path path = Paths.get("C:\\education\\institute\\student\\report.txt");
                    System.out.println("get.Name(0): %s", path.getName(0));
                    System.out.println ("subpath(0, 2): %s", path.subpath (0, 2));}
}
```

What is the result?

A. getName (0): C:\
subpath (0, 2): C:\education\report.txt
B. getName(0): C:\
subpth(0, 2): C:\education
C. getName(0): education
subpath (0, 2): education\institute
D. getName(0): education
subpath(0, 2): education\institute\student
E. getName(0): report.txt
subpath(0, 2): insritute\student

Answer: C

Explanation:
Example:
Path path = Paths.get("C:\\home\\joe\\foo");
getName(0)
-> home
subpath(0,2)
Reference: The Java Tutorial, Path Operations

Question: 11

To provide meaningful output for:
System.out.print(new Item ()):
A method with which signature should be added to the Item class?

A. public String asString()
B. public Object asString()
C. public Item asString()
D. public String toString()
E. public object toString()
F. public Item toString()

Answer: D

Explanation:
Implementing toString method in java is done by overriding the Object's toString method. The java toString() method is used when we need a string representation of an object. It is defined in Object class. This method can be overridden to customize the String representation of the Object.

Note:
Below is an example shown of Overriding the default Object toString() method. The toString() method must be descriptive and should generally cover all the contents of the object.

```
class PointCoordinates {
        private int x, y;
        public PointCoordinates(int x, int y) {
                this.x = x;
                this.y = y;
        }
        public int getX() {
                return x;
        }
        public int getY() {
                return y;
        }
        // Custom toString() Method.
        public String toString() {
                return "X=" + x + " " + "Y=" + y;
        }
}
```

Question: 12

Given the code fragment:

```
public class DisplaValues {
        public void printNums (int [] nums){
                for (int number: nums) {
                        System.err.println(number);
}
        }
}
```

Assume the method printNums is passed a valid array containing data. Why is this method not producing output on the console?

A. There is a compilation error.
B. There is a runtime exception.
C. The variable number is not initialized.
D. Standard error is mapped to another destination.

Answer: D

Explanation:
The code compiles fine.
The code runs fine.
The err stream can be redirected.
Note:
System.out.println -> Sends the output to a standard output stream. Generally monitor.
System.err.println -> Sends the output to a standard error stream. Generally monitor.

err is the "standard" error output stream. This stream is already open and ready to accept output data.
Typically this stream corresponds to display output or another output destination specified by the host environment or user. By convention, this output stream is used to display error messages or other information that should come to the immediate attention of a user even if the principal output stream, the value of the variable out, has been redirected to a file or other destination that is typically not continuously monitored.
Reference: java.lang.System

Question: 13

Which method would you supply to a class implementing the Callable interface?

A. callable ()
B. executable ()
C. call ()
D. run ()
E. start ()

Answer: C

Explanation:
public interface Callable<V>
A task that returns a result and may throw an exception. Implementors define a single method with no arguments called call.
Note:
Interface Callable<V>
Type Parameters:
V - the result type of method call
The Callable interface is similar to Runnable, in that both are designed for classes whose instances are potentially executed by another thread. A Runnable, however, does not return a result and cannot throw a checked exception.
The Executors class contains utility methods to convert from other common forms to Callable classes.
Reference: java.util.concurrent

Question: 14

Given the existing destination file, a source file only 1000 bytes long, and the code fragment:

```
public void process (String source, String destination) {
try (InputStream fis = new FileInputStream(source);
OutputStream fos = new FileOutputStream(destination)
) {
byte [] buff = new byte[2014];
int i;
while ((i = fis.read(buff)) != -1) {
fos.write(buff,0,i); // line ***
}
```

```
} catch (IOException e) {
System.out.println(e.getClass());
}
}
```

What is the result?

A. Overrides the content of the destination file with the source file content
B. Appends the content of the source file to the destination file after a new line
C. Appends the content of the source file to the destination file without a break in the flow
D. Throws a runtime exception at line ***

Answer: A

Explanation:
The whole of the FileInputStream will be read (see ** below).
The content of the FileInputStream will overwrite the destination file (see *** below).
* A FileInputStream obtains input bytes from a file in a file system. What files are available depends on the host environment.
FileInputStream is meant for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.
** FileInputStream.read (byte[] b)
Reads up to b.length bytes of data from this input stream into an array of bytes.
Parameters:
b - the buffer into which the data is read.
Returns:
the total number of bytes read into the buffer, or -1 if there is no more data because the end of the file has been reached.
*** FileOutputStream
You can construct a FileOutputStream object by passing a string containing a path name or a File object.
You can also specify whether you want to append the output to an existing file.
public FileOutputStream (String path)
public FileOutputStream (String path, boolean append)
public FileOutputStream (File file)
public FileOutputStream (File file, boolean append)
With the first and third constructors, if a file by the specified name already exists, the file will be overwritten. To append to an existing file, pass true to the second or fourth constructor.
Reference: Class FileInputStream
Reference: Class FileOutputStream

Question: 15

Which two codes correctly represent a standard language locale code?

A. ES
B. FR
C. U8
D. Es

E. fr
F. u8

Answer: A, D

Explanation:
Language codes are defined by ISO 639, an international standard that assigns two- and three-letter codes to most languages of the world.
Locale uses the two-letter codes to identify the target language.
ES, Es: Spanish
Incorrect answers:
FR: This is a language code for France.
fr: This is a language code for France.
u8, U8: No such language codes.
Reference: IS0 639

Thank You for trying 1Z0-804 PDF Demo

To Buy Latest 1Z0-804 Full Version Download visit link below

https://www.certkillers.net/Exam/1Z0-804

Start Your 1Z0-804 Preparation

[Limited Time Offer] Use Coupon "CKNET" for Further discount on your purchase. Test your 1Z0-804 preparation with actual exam questions.